SNO-STR-91-077

It's Alive! - The Birth of SNOMAN

M.D. Lay, D.L. Wark, N. West
University of Oxford

December 4, 1991

1 Introduction

This document describes the work currently proceeding at Oxford on the Sudbury Neutrino Observatory Monte carlo and **AN**alysis code, or **SNOMAN**. Our goal has been to incorporate the design objectives and decisions reached at earlier SNO Software Group meetings into an actual running piece of code. The hope was that this code could then form a basis for the collaboration's distributed software development by serving as a template into which new routines could be plugged for testing and development. This hope lead us to make one change to the basic structure of the software as it had been earlier discussed. In order to increase the autonomy of individual routines (and thus simplify their writing by different authors at different institutions) we wanted to develop a program stucture where routines interacted only with a central data structure and, to the extent possible, not with each other. It was clear that combining program flow control and physics calculations within individual routines would greatly complicate this task. We therefore divided the software into two types of routines, software control routines and physics "processors", and from these we developed the structure for SNOMAN which will be described more fully in the next section of this report. Further sections then describe the prototype version of SNOMAN (which is already in excess of 2000 lines, not counting EGS4) which we are currently running at Oxford, and our proposed plans to further develop and distribute this code. To summarize, the key elements behind the design of SNOMAN are:

- A central ZEBRA data structure describing the current event.
- A set of independent "Processors" each of which performs a single operation on a single point in the data structure and updates the data structure. Processors describe the detector characteristics and physics.
- A set of general control routines that scan the data structure and call the appropriate processor.

The advantages of such a design are:

- Parallel Development of Processors.
  Once the central data structure has been designed software development on the various processors can proceed in parallel as they only interface to this structure NOT to other processors.
- Smooth Development Cycle.
  Initially each processor can be more or less a dummy and each can then be refined as required. There is never the need for a major rewrite to introduce a new feature.
- Access to All Data.
  As the central data structure describes the complete event including the tracking of particles, any analysis routine can look at any quantity; there is never any need to dig a local variable out of another processor!
- Maintainable and Understandable.
  Separating out control from processing leads to a simple well structured program that should be easy to maintain and understand.

2 The Structure of SNOMAN

The data structure for SNOMAN is rooted in the concept of a ZEBRA bank. ZEBRA is a CERN supplied memory manager with a transparent structure and an absolutely opaque manual. Briefly, a bank consists of a bank pointer to indicate its position within an extended block of allocated memory, a set of links which point to related banks, and a set of data words which contain information about the particular object which the bank represents. Both the links and the data words are accessed using defined offsets which give their address in memory relative to the bank pointer (for more details, see the CERN ZEBRA manual, and may the Lord have mercy on your soul). The existing banks used by SNOMAN and their relationships are shown in appendix I, which also shows the contents of the banks.

An event in SNOMAN then consists of a linked group of banks which the control codes produce in order (a diagram of the routines mentioned below is given in appendix II). First a set of routines initialize the whole code. Then the main event loop routine EVMAIN generates event banks either by reading real events from tape or generating them by Monte Carlo (or possibly reading in old Monte Carlo events from tape). The Monte Carlo consists of a routine which generates source vertices (which represent the initial neutrino interaction, or a relevant background process). The rest of the generation process then consists of iterative calls to vertex and track processors. A vertex is generated anytime anything happens to a particle (which is defined broadly, for instance, a neutron, electron, or Čerenkov photon). The vertex is linked to an incoming track and an outgoing track. A track represents the transport of a particle within a single medium. Vertices come in four types, either sources, interactions, boundaries, or sinks. The control routine looks at a vertex, and

given the type it represents it then calls the appropriate vertex processor which adds an outgoing track. The control routine then calls the track processor appropriate to a particular track to decide what kind of vertex will be at its end, which is then generated. When a track reaches the PMT boundary, a processor is called to determine if a PMT hit has been generated and, if so, to produce the appropriate PMT banks. (Of course, in the case of real data, the control routine fills the PMT banks directly from the tape). After all tracks end in sink vertices, the fitter is now called to reconstruct the event and produce a fitter bank.

As the code currently functions there is now for each event a call to the event analysis routine (which packs whatever information is desired into HBOOK4 ntuples for later analysis with PAW) and the event view routine (which will do the graphics and currently does nothing). The banks for the individual event are then wiped and the process restarted. It may be desirable (ie, faster) to create and save all the banks for all the events before calling these routines. This might allow the user to interactively decide what results from each event to look at without the necessity of re-generating the events if he decides a different paramenter is interesting, however it will create a considerable expense in memory and perhaps paging time. The most desirable solution may be hardware dependent, and thus may change during the $\approx$10 year lifetime of this code. One of the advantages of the highly modular structure we are trying to build is that this basic change in the operation of the code can be accomplished through changing about ten lines of code, thus we are free to decide later. It is our intent that all routines in the code can be easily swapped out and changed, control routines as well as processors.

3 SNOMAN prototype: Objectives, Progress, and Performance

The primary objectives were to develop a basic data structure in ZEBRA, set up the control routines and to see what the memory and CPU requirements were. This requires a geometry to define the boundaries so that the requisite number of vertices and tracks are generated for each event, even though the physical processes each represents are often ignored. One concern was the memory; each ZEBRA bank has a 10 word (40byte) overhead.

The development was organised into version numbers. At the time of writing there have been 3 versions:-

0.00	Source:	Cerenkov photons traveling radially out from centre.
	Geometry:	Spherically symmetric 4 volume: D2O, Acrylic, H2O, PMT.
	Media:	All transparent; no refraction or reflection.
	Detection:	Dummy.
	Fitting:	Dummy.
0.01	Source:	Monoenergetic electron randomly within 500cm of centre.
	Geometry:	Spherically symmetric 4 volume: D2O, Acrylic, H2O, PMT.

	Media:	All transparent; no refraction or reflection.
	Detection:	Black disc PMTs arranged in constant azimuthal rings.
	Fitting:	Minimise time chi-squared assuming straight line tracks.
0.02	Source:	Monoenergetic electron randomly within 500cm of centre.
	Geometry:	Spherically symmetric 4 volume: D2O, Acrylic, H2O, PMT.
	Media:	All transparent; no refraction or reflection.
	Detection:	PMT's with correct angular response.
	Fitting:	Minimise time chi-squared assuming straight line tracks. Version 0.02 also contains some rationalization of the control structure.

A few output histograms from version 0.02 are shown in the Figures. The performance has been studied on a VAX 4000 using the performance analyser PCA. Some timings were repeated on the APOLLO DN1000 giving a CPU ratio:

APOLLO CPU = 2.3 * VAX CPU

3.1 Memory Requirements

The memory used is approximately linear in the number of Cerenkov photons generated. The requirement is:

200 words (800 bytes) per Cerenkov photon.

3.2 CPU Requirements

The numbers given are for the APOLLO DN1000 running version 0.01:

Initialization	3.1 secs
One Event (approx 220 Cerenkov photons)	0.23 secs

The time spent processing events is divided up roughly as follows:

ZEBRA memory management	49%
Control routines	26%
Fitting	9%
Geometry	7%
EGS4 + Cerenkov photon generation	6%
PMT hit recording	3%

Note: the time is roughly linear in the number of photons produced.

As can be seen, at this stage the times are dominated by ZEBRA and the control aspects of the code. However all future development is unlikely to have any significant effect in these areas.

4 Plan for Future Development

It is our opinion that SNOMAN Version 0.02 has demonstrated the viability of this scheme, and we now intend to develop it further. In the short term, we will try to further enhance the modularity of the subroutines and minimize their interactions. For instance, in the current code if one changed over to a different fitter it would be necessary to also redo the event analysis and HBOOK initialization routines to include the different parameters that the new fitter would produce. While some such problems are inevitable, we are trying to reduce them to an absolute minimum. For instance, all processors know when they are being called for the first time, thus all processors are expected to initialize themselves on the first call. However they cannot easily know when they are being called for the last time, thus the author must supply a routine TMPROC (where the name PROC varies for each routine) which acts as an entry point to terminate the routine gracefully. We are also working on what banks will be necessary for the various interaction routines to communicate with PHINTL. We want to put a fair amount of time into thinking about the best way to do such things now, because once the code is distributed to the collaboration such conventions will quickly tend to become entrenched, making later changes painful.

This work will be in preparation for the release of Version 1.00 to the whole collaboration, hopefully sometime in the late spring. It is our intent that Version 1.00 will come complete with all routines necessary to make a completely functional bare-bones SNO Monte Carlo. In other words it will be able to generate all interesting categories of events, produce light from them, propagate it through a reasonably realistic geometry and detect it with a reasonably accurate representation of the PMT's. Thus an author developing the actual code for, say, Fresnel scattering or neutron propagation from neutral current events will be able to plug their code into a frame which is debugged and gives a least a rough idea of how the rest of the detector simulation code will react to any changes they make. Of course, to produce the code on this time scale will require many approximations to be made, but we are not promising code which produces physics results, only a skeleton to build such code on. Our current biggest worry is the PMT beta-gamma events. The concern is that ZEBRA overheads are so high that with current technology SNOMAN will be unable to simulate events any faster than nature produces them. All we can currently say about that is we are working on it and will let you know. Version 1.00 will also include a manual which gives fairly detailed information on how it is put together, and the individual routines will be commented to the point that they will be usefully self-documented (believe it or not, some of our existing code even conforms to this oft-quoted but seldom realized ideal, hopefully it all will before it is distributed).

We also need to interact with the rest of the collaboration so that the code

management process which we decide to use is in place and operating by the time that Version 1.00 is ready for distribution. It is particularly important that a referee system be instituted whereby each piece of code must be tested and approved by someone other than the author so that new routines will hopefully contain somewhat fewer bugs. We must also develop a system by which we decide, in the case of more than one routine being available, which will be used in the SNO-wide standard code.

5 Appendix I

5.1 SNOMAN: Data Structures

This shows the structural relationship between the banks; a bank indented to level n is supported by the bank that is indented to a level n-1.

EV Event bank.
. EGS4 EGS4 bank. One for each track passed to EGS4.
. VX Vertex bank.
. . TK Track bank.
. PM PMT bank. One for each PMT that fires.
. . PH PMT hit bank. One for each hit.
. FT Fitter bank. One for each event.

5.2 EV - Event bank

Description
Holds all information global to one event.
Reference Links
None.
Structural Links

-4	-KEVFT	FT	Fitter bank.
-3	-KEVPM	PM	First PM bank.
-2	-KEVVX	VX	First VX bank.
-1	-KEVEGS	EGS4	First EGS4 bank.

Status Bits Note: Any status bit parameters are defined to work with bit functions such as IAND and BTEST.

Data Words
All data words are integer.

+1	KEVRUN	Run Number.
+2	KEVEVN	Event Number.
+3	KEVRTP	Run_Type_Code (see Run Type codes).
+4	KEVDTE	Date.
+5	KEVTIM	Time (seconds past midnight).
+6	KEVNSC	Nano-sec time (since last completed second).
+7	KEVNPM	Number of PMTs that fired.
+8	KEVRNS	Initial random number seed (MC events only).

5.3 EGS4 - EGS4 bank. One for each track passed to EGS4

Description

Records all required information returned from one call to EGS4 for a gamma or electron. Not all tracks and vertices generated by EGS4 are stored in the data structure. The convention used is that the electron or gamma passed to EGS4 is a track of zero length and ends at an EGS4 vertex. For each Cerenkov photon generated an outgoing electron track is produced that terminates at a Cerenkov photon vertex.

Reference Links

-1	-KEVVX	VX	EGS4 vertex

Structural Links

None.

Status Bits

Note: Any status bit parameters are defined to work with bit functions such as IAND and BTEST.

Data Words

All data words are floating point.

+1	KE4NCE	Number of Cerenkov photons generated
+2	KE4EDP	Total Energy deposited in Medium 1
+3	KE4RNG	Total Distance traveled by the electron
+4	KE4NST	Number of steps within EGS4

5.4 VX - Vertex bank

Description

There is one VX bank for each "event" in the life of a particle; the VX represents a state transition. See also the EGS4 bank. VX banks are assigned to groups; all members of a group are linked together via reference links. The only purpose of this grouping is to speed up processing e.g. to ignore Cerenkov photon creation vertices when scanning for sink vertices. What constitues a groups has yet to be defined.

Reference Links

-4	-KVXVXG	VX	Next member of same group.
-3	-KVXPH	PH	If incoming track produces a PMT hit.
-2	-KVXTKI	TK	Of incoming track (or 0 if vertex class is Source).

Structural Links

-1	-KVXTK	TK	First outgoing track (or 0 if vertex class is Sink)

Status Bits

Note: Any status bit parameters are defined to work with bit functions such as IAND and BTEST.

1	KVXSRC	Source

2	KVXBOU	Boundary
3	KVXINT	Interaction
4	KVXSNK	Sink

Note: more than one of the above may be set, e.g. a boundary may also prove to be a sink. If the bit is set it implies that the VX has been processed by the appropriate vertex routine.

Data Words

All data words are floating point.

+1	KVXCLS	Class: = 1 Source, = 2 Boundary, = 3 Interaction, = 4 Sink For a VX that has more than one of the status bits corresponding to these classes, the class will be the highest one.
+2	KVXINC	Interaction_Code (see Interaction codes)
+3	KVXPSX	Position X.
+4	KVXPSY	Position Y.
+5	KVXPSZ	Position Z.
+6	KVXTIM	Time in secs since event time (in EV).
+7	KVXIDM	First Media_Code: medium of incoming track. See section 1.
+8	KVXIM2	Second Media_Code (= First medium if not boundary)
+9	KVXBNX	Boundary normal X, (only defined if Boundary status bit set)
+10	KVXBNY	Boundary normal Y, (")
+11	KVXBNZ	Boundary normal Z, (")

5.5 TK - Track bank

Description

Describes the transport of one particle between its start and end vertices. Formally the information in the TK bank refers to the particle at its end vertex. For most tracks the quantities will be invarient along the track but one exception is that EGS4 will produce electron tracks at the point at which they interact to produce Cerenkov photons. The direction of such a track will not correspond to a straight line joining its start and end vertices.

Reference Links

-1	-KTKVX	VX	Track end vertex.

Structural Links

None.

Status Bits

Note: Any status bit parameters are defined to work with bit functions such as IAND and BTEST.

Data Words

All data words are floating point.

+1	KTKIDP	Particle_ID_Code (see Particle codes).

+2	KTKDRX	Direction cosine X.
+3	KTKDRY	Direction cosine Y.
+4	KTKDRZ	Direction cosine Z.
+5	KTKENE	Energy.
+6	KTKIDM	Media_Code (see Media codes)
+7	KTKPL1	Polarisation 1.
+8	KTKPL2	Polarisation 2.

5.6 PM - PMT bank. One for each PMT that fires

Description

One bank for each PMT that fires. The hits described in the PH banks may not all be resolvable. The PM bank holds the combined result of all PHs.

Reference Links

None.

Structural Links

-1	-KPMPH	PH	First PMT hit.

Status Bits

Note: Any status bit parameters are defined to work with bit functions such as IAND and BTEST.

Data Words

All data words are floating point.

+1	KPMNUM	PMT number (Give access to PMT tables)
+2	KPMPSX	Position X.
+3	KPMPSY	Position Y.
+4	KPMPSZ	Position Z.
+5	KPMETM	Earliest time of any hit in the PH chain of hits.
+6	KPM...	Description of the total PMT signal. To be defined but must be capable of holding real data as well as MC.

5.7 PH - PMT hit bank. One for each hit

Description

One for each hit on PMT.

Reference Links

-1	-KPHVX	VX	VX whose incoming track produced hit.

Structural Links

None.

Status Bits

Note: Any status bit parameters are defined to work with bit functions such as IAND and BTEST.

Data Words

All data words are floating point.

+1	KPHTIM	Time.
+2	KPHTYP	Pulse type: =1 clean, = 2 noise, = 3 pre-pulse, = 4 after-puls
+3	KPHHEI	Pulse height.

5.8 FT - Fitter bank. One for each event

Description

One bank generated for each event by fitter

Reference Links

Structural Links

None.

Status Bits

Note: Any status bit parameters are defined to work with bit functions such as IAND and BTEST.

Data Words

All data words are floating point.

+1	KFTXE	Fit value of the x coordinate
+2	KFTYE	Fit value of the y coordinate
+3	KFTZE	Fit value of the z coordinate
+4	KFTTE	Fit value of the event time
+5	KFTDXE	Error on the x coordinate
+6	KFTDYE	Error on the y coordinate
+7	KFTDZE	Error on the z coordinate
+8	KFTDTE	Error on the event time
+9	KFTIPM	Number of PMT's hit
+10	KFTICT	Number of interations of MRQMIN
+11	KFTCHI	Final fit Chi**2

6 Appendix II

6.1 SNOMAN: Software Structure

The major routines called in the current SNOMAN structure are:

```
MAIN
. INMAIN
. . INZEBR
. . INHBK
. . INPMT
. EVMAIN
. . EVREAD
. . . READEX
. . . READMC
. . MCMAIN
. . . MCGENR
. . . MCEVLV
. . . . VXMAIN
. . . . . VXSRC
. . . . . VXBOU
. . . . . . VXNULL
. . . . . VXINT
. . . . . . VXEGS4
. . . . . . . HATCH
. . . . . . . EGSINP
. . . . . . . SHOWER
. . . . . . . EGSOUT
. . . . . . . . TKNULL
. . . . . . . . VXNULL
. . . . . . VXCERN
. . . . . . VXNULL
. . . . . VXSNK
. . . . . . VXNULL
. . . . TKMAIN
. . . . . GEBOU
. . . . . PHINTL
. . . . . TKNULL
. . EVFITR
. . . MRQMIN
. . . . MRQCOF
. . EVVIEW
. . EVANAL
. TMMAIN
. . TMHBK
. . TMZEBR
```

which are:

MAIN	Main routine.
INMAIN	Initialisation main routine.
INZEBR	Initialise ZEBRA and bank definitions.
INHBK	Initialise HBOOK.
INPMT	Initialise PMT hit routine in VXSNK.
EVMAIN	Event loop main routine.
EVREAD	Read experiment or MC data file.
READEX	Read experiment data file.
READMC	Read MC data file.
MCMAIN	MC event generation.
MCGENR	MC Event generation (EV + seed VXs and TKs)
MCEVLV	MC Event evolution.
VXMAIN	VX processor main routine.
VXSRC	VX processor for source vertices.
VXBOU	VX processor for boundary vertices.
VXINT	VX processor for interaction vertices.
VXEGS4	VX processor for EGS4 tracks.
HATCH	EGS4 initialisation.
EGSINP	Load EGS4 with current track.
SHOWER	EGS4 track processing.
EGSOUT	Add Cerenkov interaction to event D/S.
VXCERN	VX processor for Cerenkov photon production.
VXSNK	VX processor for sink vertices, PMT's
TKMAIN	TK processor main routine.
GEBOU	Geometry: Distance to next boundary.
PHINTL	Physics: Interaction length.
EVFITR	Event reconstruction.
MRQMIN	Levenberg-Marquart minimising main routine.
MRQCOF	Levenberg-Marquart minimising step.
EVVIEW	Event display.
EVANAL	Event analysis.
TMMAIN	Termination main routine.
TMHBK	Terminate HBOOK.
TMZEBR	Terminate ZEBRA.

The following are called from more than one routine:
TKNULL Null TK processor (propagate to null interaction)
VXNULL Null VX processor (outgoing track = incoming track)

These routines are very useful as they handle all the ZEBRA code needed to create the banks with the appropriate links. They can then be used to create banks whose data words can then be filled by other routines.

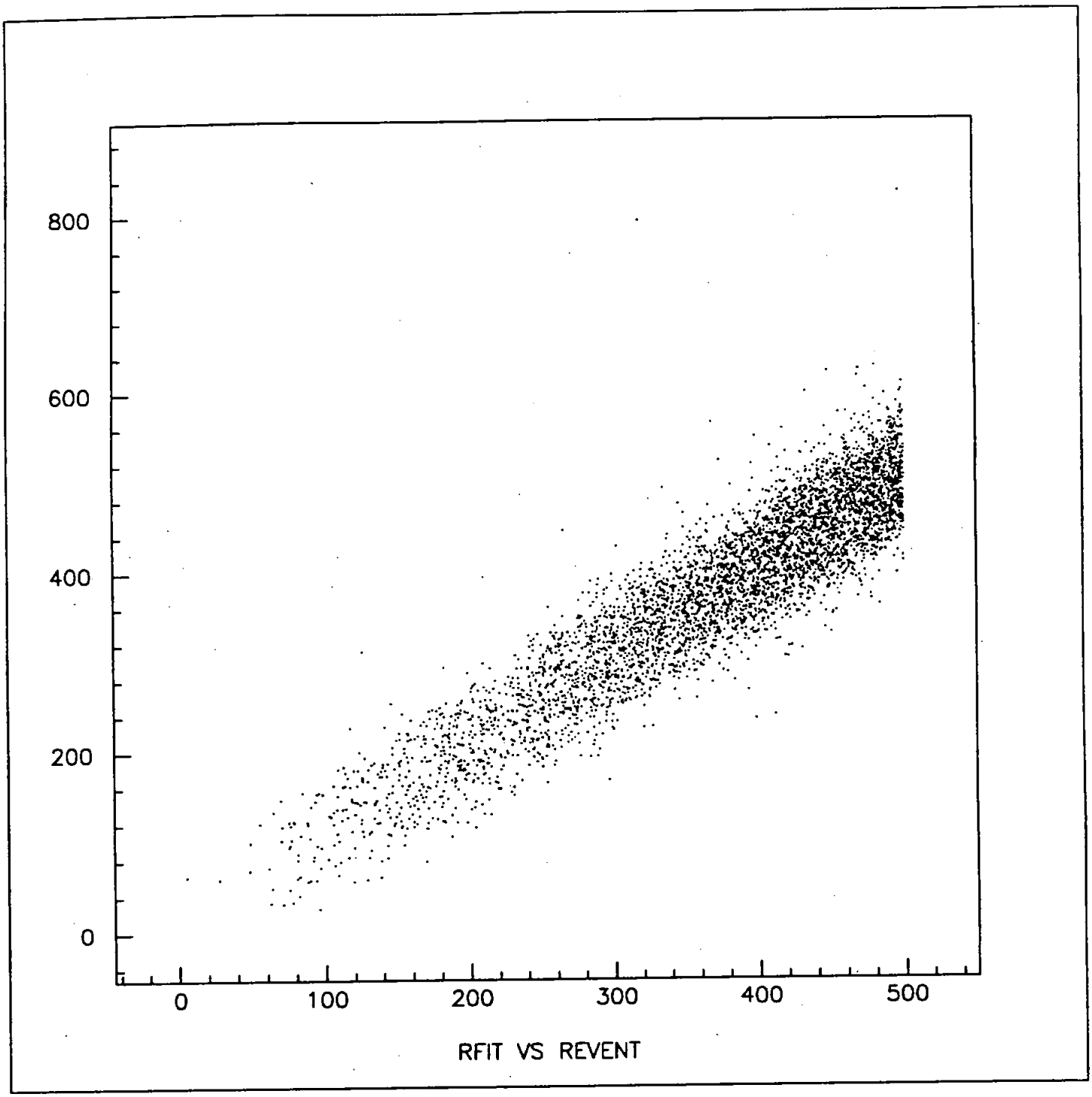

Fig. 1 Scatter plot of r_{fit} vs. r_{event}.

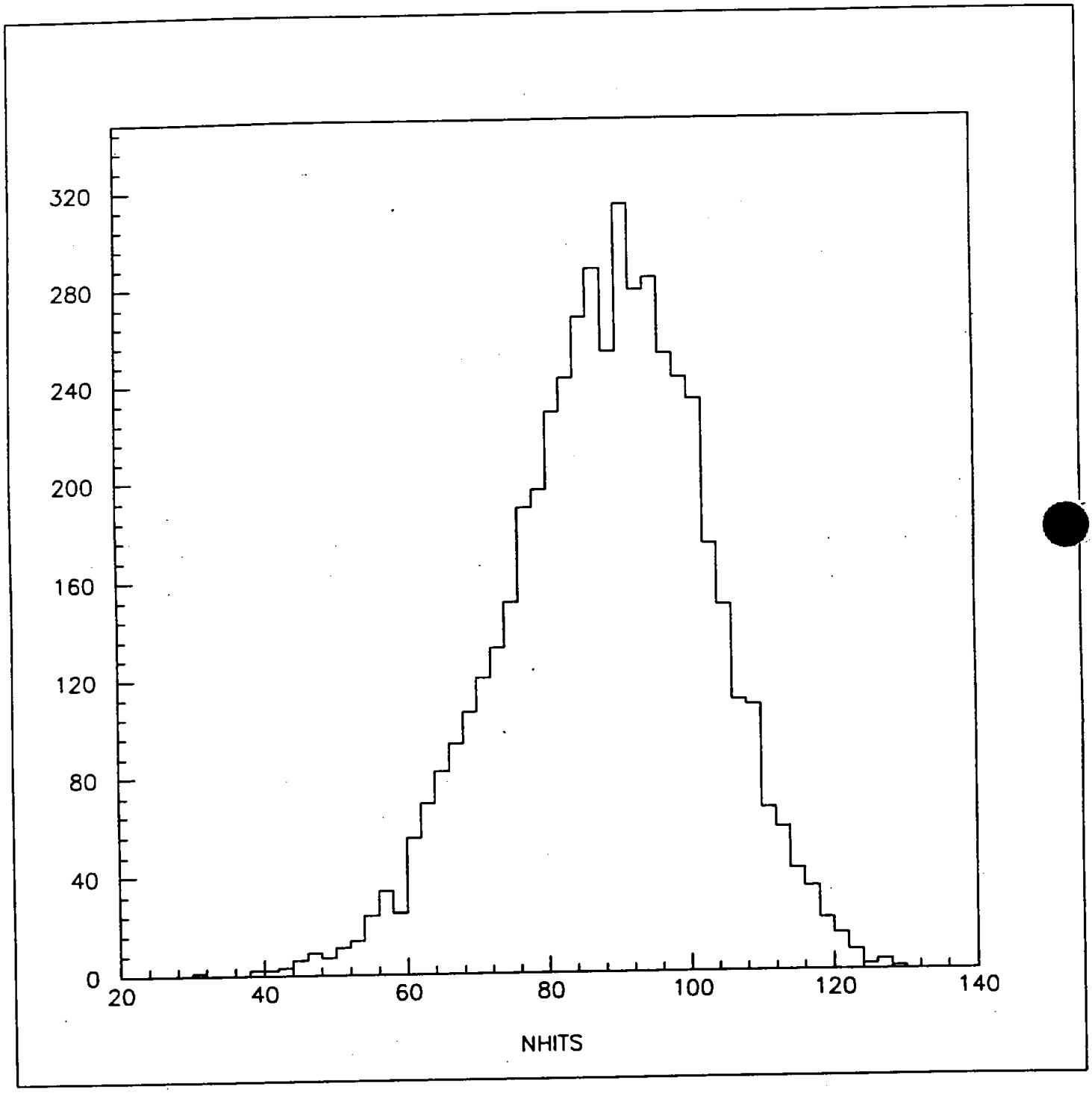

Fig. 2. Histogram of number of hits.

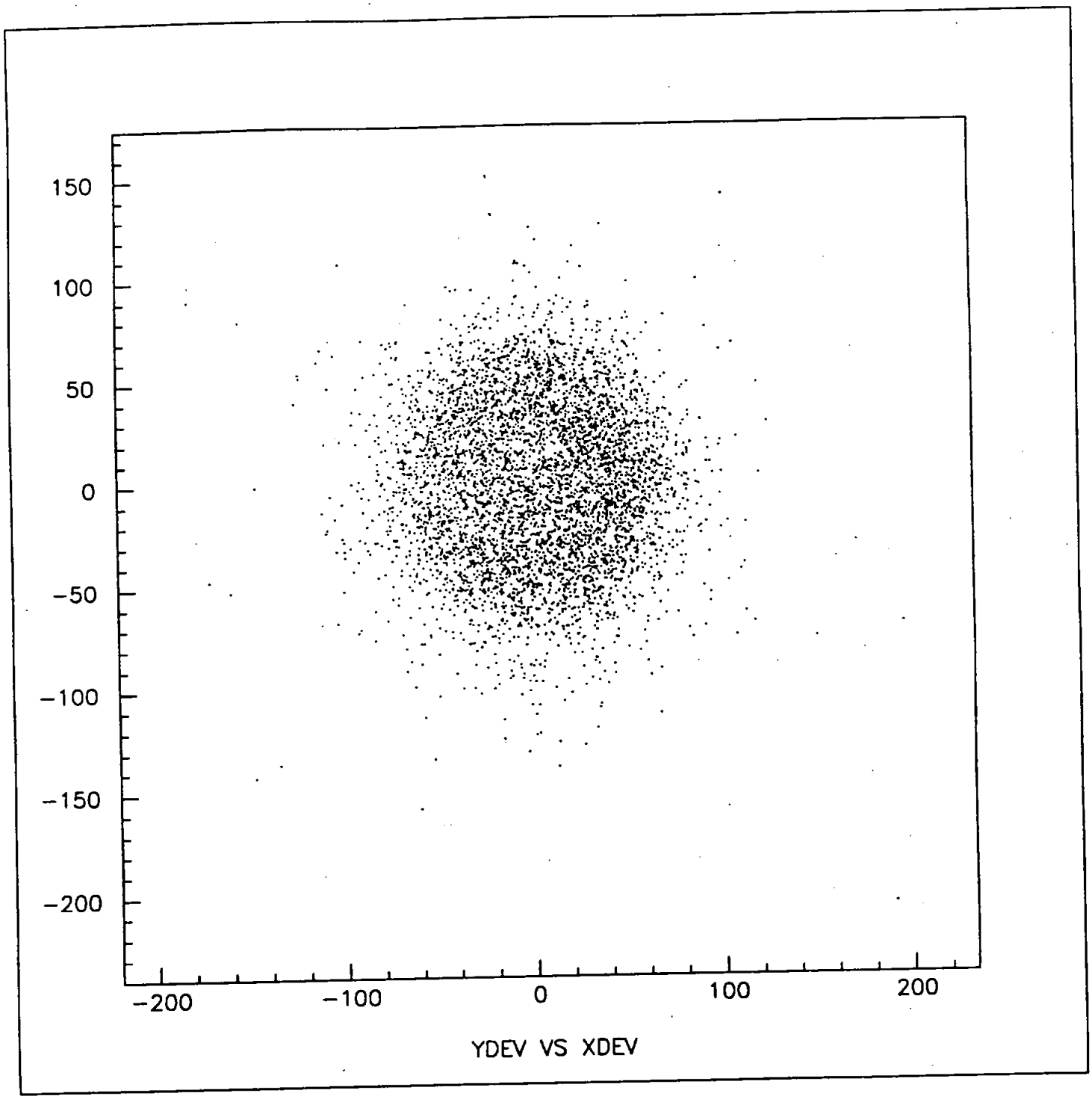

Fig. 3. Scatter plot of $(Y_{fit} - Y_{event})$ vs. $(X_{fit} - X_{event})$